Secure Multi-Party Computation
A Short Tutorial
By no means a survey!

Manoj Prabhakaran :: University of Illinois at Urbana-Champaign
Secure Multi-Party Computation
A Short Tutorial
Part I
Can we have an auction without an auctioneer?!

- Declared winning bid should be correct
- Only the winner and winning bid should be revealed
Using data without sharing?

Hospitals which can’t share their patient records with anyone

But want to data-mine on combined data
A general problem

To compute a function of private inputs without revealing information about the inputs

Beyond what is revealed by the function
Poker With No Dealer?

Need to ensure:
- Cards are shuffled and dealt correctly
- Complete secrecy
- No “cheating” by players, even if they collude
- No universally trusted dealer
The Ambitious Goal

Without any trusted party, securely do
- Distributed Data mining
- E-commerce
- Network Games
- E-voting
- Secure function evaluation
- ....

Any Task!
Emulating Trusted Computation

- Encryption/Authentication allow us to emulate a trusted channel
- Secure MPC: to emulate a source of trusted computation
- Trusted means it will not “leak” a party’s information to others
- And it will not cheat in the computation
Security Issues to Consider

- Protocol may leak a party’s secrets
  - Clearly an issue
  - Even if we trust everyone not to cheat in our protocol (i.e., honest-but-curious)
  - Also, a liability for a party if extra information reaches it (e.g., in medical data mining)

- Protocol may give adversary illegitimate influence on the outcome
  - Say in poker, if adversary can influence hands dealt
Defining Security

- REAL/IDEAL paradigm

Security guarantee: Whatever an adversary can do in the REAL world, an adversary could have done the same in the IDEAL world

- Can’t blame the protocol for anything undesirable

- Will return to this in Part II
An example

An auction, with Alice and Bob bidding

Rules:

- A bid is an integer in the range [0, 100]
- Alice can bid only even integers and Bob odd integers
- Person with the higher bid wins

Goal: find out the winning bid (winner & amount) without revealing anything more about the losing bid (beyond what is revealed by the winning bid)
An example

Secure protocol:

- Count down from 100
- At each even round Alice announces whether her bid equals the current count; at each odd round Bob does the same
- Stop if a party says yes

Dutch flower auction
Adversary

- REAL-adversary can corrupt any set of players
- IDEAL-adversary should corrupt the same set of players
- More sophisticated notion: adaptive adversary which corrupts players dynamically during/after the execution
- We’ll stick to static adversaries

Passive vs. Active adversary: Passive adversary gets only read access to the internal state of the corrupted players. Active adversary overwrites their state and program.
Functionality

- Functionality: program of the trusted party to be emulated
- Can consider arbitrary (efficient) functionalities
- An important class: Secure Function Evaluation (SFE)
  - e.g. Oblivious Transfer (coming up)
- Can be randomized: e.g. Coin-tossing
- “Reactive” functionalities (maintains state over multiple rounds)
  - e.g. Secure Poker
Oblivious Transfer

- Pick one out of two, without revealing which

- Intuitive property: transfer partial information “obliviously”

$$x_0, x_1, b, x_b$$

We Predict STOCKS!!

A: up, B: down

IDEAL World

I need just one
But can’t tell you which

Sure

All 2 of them!
Oblivious Transfer

Pick one out of two,

<table>
<thead>
<tr>
<th>x₀</th>
<th>x₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>( )</td>
</tr>
</tbody>
</table>

We Predict STOCKS!!

A: up, B: down

I need just one. But can’t tell you which.

Sure
Can we REAL-ize them?

- Are there protocols which securely realize these functionalities?

- Turns out the REAL/IDEAL definition can be “too strong” for active corruption
  - Unless carefully formulated... (later)

- Will start with passive security (i.e., security against passive corruption)
An OT Protocol (passive corruption)

Using a (special) encryption

PKE in which one can sample a public-key without knowing secret-key

$c_{1-b}$ inscrutable to a passive corrupt receiver

Sender learns nothing about $b$
1-out-of-N OT

\[ f( (x_1, \ldots, x_N); i ) = (\bot; x_i) \]

For passive security: simply run N copies of 1-out-of-2 OT, with inputs for \( j^{th} \) instance being \((0, x_j; b_j) \) where \( b_j = 1 \) iff \( j=i \)

Aside: active security easily achieveable too using a randomized protocol using N-1 copies of 1-out-of-2 OT
### Plan: General SFE

<table>
<thead>
<tr>
<th>Passive</th>
<th>Unlimited Corruption</th>
<th>Honest Majority</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>“GMW” protocol</td>
<td>BGW protocol</td>
</tr>
<tr>
<td>Active</td>
<td>via AMD circuits</td>
<td>via error-correcting secret-sharing</td>
</tr>
</tbody>
</table>

via FMW circuits
2-Party SFE

Secure Function Evaluation (SFE) IDEAL:

Trusted party takes \((X;Y)\). Outputs \(g(X;Y)\) to Alice, \(f(X;Y)\) to Bob

Randomized Functions: \(g(X;Y;r)\) and \(f(X;Y;r)\) s.t. neither party knows \(r\) (beyond what is revealed by output)

OT is an instance of a (deterministic) 2-party SFE

\[
g(x_0,x_1;b) = \text{none}; \quad f(x_0,x_1;b) = x_b
\]

Single-Output SFE: only one party gets any output
2-Party SFE

- Can reduce any SFE (even randomized) to a single-output deterministic SFE

\[ f'(X, M, r_1; Y, r_2) = ( g(X; Y; r_1 \oplus r_2) \oplus M, f(X; Y; r_1 \oplus r_2) ). \]

- Compute \( f'(X, M, r_1; Y, r_2) \) with random \( M, r_1, r_2 \)

- Bob sends \( g(X, Y; r_1 \oplus r_2) \oplus M \) to Alice

- Passive secure

- For active security, \( f' \) authenticates (one-time MAC) as well as encrypts \( g(X; Y; r_1 \oplus r_2) \) using keys input by Alice

- Generalizes to more than 2 parties

- Can reduce any single-output deterministic SFE to OT!
“Completeness” of OT

- Can reduce any single-output deterministic SFE to OT!
- For passive security
  - Proof of concept for 2 parties: An inefficient reduction
  - Yao’s garbled circuit for 2 parties (Not covered today)
- “Basic GMW”: Information-theoretic reduction to OT
- Later: OT is complete even for active security
“Completeness” of OT: Proof of Concept

- Single-output 2-party function $f$

- Alice (who knows $x$, but not $y$) prepares a table for $f(x, \cdot)$ with $N = 2^{|y|}$ entries (one for each $y$)

- Bob uses $y$ to decide which entry in the table to pick up using 1-out-of-N OT (without learning the other entries)

- Bob learns only $f(x,y)$ (in addition to $y$). Alice learns nothing beyond $x$.

- Problem: $N$ is exponentially large in $|y|$
Functions as Circuits

- Directed acyclic graph
- Nodes: AND, OR, NOT, CONST gates, inputs, output(s)
- Edges: Boolean valued wires
- Each wire comes out of a unique gate, but a wire might fan-out
- Can evaluate wires according to a topologically sorted order of gates they come out of
Functions as Circuits

- e.g.: OR (single gate, 2 input bits, 1 bit output)
- e.g.: $X > Y$ for two bit inputs $X=x_1x_0$, $Y=y_1y_0$:
  $$(x_1 \land \neg y_1) \lor (\neg(x_1 \oplus y_1) \land (x_0 \land \neg y_0))$$
- Can directly convert a truth-table into a circuit, but circuit size exponential in input size
- Can convert any (“efficient”) program into a (“small”) circuit
- Interesting problems already given as succinct programs/circuits
Basic GMW

Adapted from the famous Goldreich-Micali-Wigderson (1987) protocol (due to Goldreich-Vainish, Haber-Micali,...)

Efficient passive secure MPC based on OT, without any other computational assumptions

Idea: Computing on secret-shared values
Computing on Shares

Fix any “secret” $s$. Let $a, b$ be random conditioned on $s = a + b$. (All elements from a finite field.)

Each of $a, b$ by itself carries no information about $s$. (e.g., can pick $a$ at random, set $b = s - a$.)

Will write $[s]_1$ and $[s]_2$ to denote shares of $s$
Computing on Shares

Let gates be + & × (XOR & AND for Boolean circuits)

Plan: shares of each wire value will be computed, with Alice holding one share and Bob the other. At the end, Alice sends her share of output wire to Bob.

\[ w = u + v \]: Each one locally computes \([w]_i = [u]_i + [v]_i\)
Computing on Shares

What about \( w = u \times v \) ?

\[
[w]_1 + [w]_2 = ( [u]_1 + [u]_2 ) \times ( [v]_1 + [v]_2 )
\]

Alice picks \([w]_1\). Can let Bob compute \([w]_2\) using the naive (proof-of-concept) protocol

Note: Bob’s input is \(([u]_2,[v]_2)\). Over the binary field, this requires a single 1-out-of-4 OT.
GMW: many parties

- m-way sharing: \( s = [s]_1 + ... + [s]_m \)
- Addition, local as before
- Multiplication: For \( w = u \times v \)
  \[ [w]_1 + ... + [w]_m = ( [u]_1 + ... + [u]_m ) \times ( [v]_1 + ... + [v]_m ) \]

- Party \( i \) computes \([u]_i[v]_i\)

- For every pair \((i,j), i \neq j\), Party \( i \) picks random \( a_{ij} \) and lets Party \( j \) securely compute \( b_{ij} \) s.t. \( a_{ij} + b_{ij} = [u]_i[v]_j \) using the naive protocol (a single 1-out-of-2 OT)

- Party \( i \) sets \([w]_i = [u]_i[v]_i + \Sigma_j ( a_{ij} + b_{ji} )\)

Allows security against arbitrary number of corruptions
GMW: with active corruption

Original GMW approach: Use Zero Knowledge proofs to force the parties to run the protocol honestly

- Needs (passive-secure) OT to be implemented using a protocol

Kilian/IPS: Direct information-theoretic reduction to OT

Alternate construction: information-theoretic reduction to OT, starting from passive-secure GMW
Passive-Secure GMW: Closer Look

Multiplication: \([w_1] + [w_2] = ( [u_1] + [u_2] ) \times ( [v_1] + [v_2] )\)

Computing shares \(a_{12}, b_{12}\) s.t. \(a_{12} + b_{12} = [u_1] \cdot [v_2] \):

- Alice picks \(a_{12}\) and sends \((-a_{12}, [u_1] - a_{12})\) to OT. Bob sends \([v_2]\) to OT.
- What if Alice sends arbitrary \((x, y)\) to OT? Effectively, setting \(a_{12} = -x, [u_1]^\prime = y - x\).
- What Bob sends to OT is \([v_2]^\prime\).

i.e., arbitrary behavior of Alice & Bob while sharing \([u_1] \cdot [v_2]\) correspond to them locally changing their shares \([u_1]\) and \([v_2]\)
Passive-Secure GMW: Closer Look

Multiplication: \[ w_1 + w_2 = (u_1 + u_2) \times (v_1 + v_2) \]

Arbitrary behavior of Alice while sharing \([u]_1 \cdot [v]_2\) and \([u]_2 \cdot [v]_1\) corresponds to her locally changing her shares of \(u\) and \(v\).

Alice changing her share from \([u]_1\) to \([u]_1'\) is effectively changing \(u\) to \(u + \Delta_u\), where \(\Delta_u = [u]_1' - [u]_1\) depends only on her own view.

Over all effect: a corrupt party can arbitrarily add \(\Delta_u\) and \(\Delta_v\) to wires \(u\) and \(v\) before multiplication.

Also, can add deltas to all input and output wires.
Active-Secure Variant of Basic GMW

Any active attack on Basic GMW protocol corresponds to an additive attack on the wires of the circuit.

Idea: “Compile” the circuit such that any additive attack amounts to error (w.h.p.), resulting in random output.

Additive Manipulation Detecting (AMD) circuits

Extension of AMD codes

e.g. encode $x$ as a vector $(x, r, xr)$ where $r$ is random from a large field. Additive attacks (without knowing $r$) detected unless $(x+\delta_1)(r+\delta_2) = (xr+\delta_3)$: i.e., $\delta_1 \cdot r + x \cdot \delta_2 + \delta_1 \cdot \delta_2 = \delta_3$. Unlikely unless $\delta_1 = 0$. 
Honest Majority

- So far, arbitrary number of parties can be corrupted (in particular, secure 2-party computation, when one party is corrupt)
  - But needed to rely on OT

- Up Next: Adversary can corrupt any set of less than \( t \) parties out of \( m \) parties (e.g., \( t = n/2, t=n/3 \))
  - Then, can get (UC) security just from secure communication channels

- Bonus: Can ask for guaranteed output delivery, a.k.a. full-security
Honest Majority

- Can’t tolerate (passive) corruption of n/2 parties unless functionality (passive) trivial for 2-party
- Can’t tolerate active corruption of n/3 parties (even for “broadcast”) if full security needed
No Fully Secure Broadcast with 1/3 Corrupt

Broadcast: A bit sent by one party to the others.

If sender honest, all honest parties should output the bit it sends (can’t abort)

All honest parties should agree on the outcome (can’t have some output 0 and others 1)
Fully Secure Broadcast with < 1/3 Corrupt

Broadcast: A bit sent by one party to the others.

If sender honest, all honest parties should output the bit it sends (can’t abort)

All honest parties should agree on the outcome (can’t have some output 0 and others 1)

There are fully secure broadcast protocols for < 1/3 corruption

We will take broadcast as an IDEAL primitive
Honest Majority

- Can’t tolerate (passive) corruption of n/2 parties unless functionality (passive) trivial for 2-party
- Can’t tolerate active corruption of n/3 parties (even for “broadcast”) if guaranteed output delivery needed
- More generally, fully secure general secure function evaluation not possible if:
  - set of parties can be partitioned into three sets, $S_1 \cup S_2 \cup S_3$ such that $S_1$, $S_2$ (separately) may be passively corrupt, and $S_3$ may be actively corrupt
- Plan: Given broadcast, get (full) security against <n/3 corruption
BGW: Passive Security

- Again, gate-by-gate evaluation of shared wire-values

- Idea 1: Use a linear secret-sharing scheme that allows local multiplication, but resulting in shares in a different linear secret-sharing scheme.
  - Need privacy only against < n/2 corruption

- Idea 2: Can move from one linear secret-sharing scheme to another securely.
Idea 1: Use a linear secret-sharing that allows local multiplication, but resulting in shares in a different linear secret-sharing scheme

**Shamir secret-sharing** using degree \( \lfloor \frac{n-1}{2} \rfloor \) polynomials (privacy against \(< \frac{n}{2} \) (\( \leq \) degree+1) corruption)

\[ [s]_i = \sigma(x_i) \text{ where polynomial } \sigma \text{ s.t. } \sigma(0) = s \]

\[ \sigma(0) = \text{ a linear combination of degree+1 shares } \{\sigma(x_i)\}_i \]

Multiplying two such polynomials for secrets \( s, t \): \( \pi = \sigma \cdot \tau \). Then \( [s \cdot t]_i = \pi(x_i) = \sigma(x_i) \cdot \tau(x_i) \) and \( \pi(0) = s \cdot t \)

Degree of \( \pi \leq n-1 \): \( \pi(0) \) reconstructible from \( n \) shares
BGW

Idea 2: Can move from linear secret-sharing scheme A to linear secret-sharing scheme B securely

- Given shares \((a_1, \ldots, a_n) \leftarrow \text{Share}_A(s)\)
- Share each \(a_i\) using scheme B: \((b_{i1}, \ldots, b_{in}) \leftarrow \text{Share}_B(a_i)\)
- Locally each party \(j\) reconstructs using scheme A: \(b_j \leftarrow \text{Recon}_A(b_{1j}, \ldots, b_{nj})\)
- Claim: \(\text{Recon}_B(b_1, \ldots, b_n) = s\)
- For any linear \(f\), \(\text{Recon}_B( f(\text{Share}_B(\overline{a})) ) = f(\overline{a})\)
- \(\text{Recon}_A\) is a linear function
Active Security

Active security with abort:

- Run (passive-secure) BGW on an AMD circuit of the function
- Each party will accept the output only if the output verifies

In IDEAL, adversary can cause selective abort, after seeing its own output

Next level of security: Fairness

In IDEAL, adversary can cause abort for all parties without knowing its own outputs
Error-Correcting Secret-Sharing

- Allows reconstruction as long as a majority of the shares submitted are correct.
- E.g., mutually authenticating shares (using statistical MACs).
- To reconstruct, look for a clique of size $n/2$ of mutually consistent shares.
Fair Honest-Majority MPC

- Share inputs using ECSS
- Run unfair protocol to obtain ECSS shares of output
- If no abort, each honest party broadcasts OK
- If all say OK, then send ECSS shares for reconstruction

Adversary who corrupts < n/2 parties can cause abort for all parties, but without knowing its own outputs

Next Level of Security: Guaranteed output delivery (a.k.a. full security)

Note: above, required broadcast to be fully secure
Full Security

The main difficulty, compared to active-secure MPC, is in identifying who cheated.

Not possible to exactly identify one cheating party.

- e.g., $[P_1 \text{ sends garbage to } P_2 \text{ over a private link }] \equiv [P_2 \text{ discards what } P_1 \text{ sent, replacing it with garbage }]

Can hope to identify a set of 2 parties, at least one of which is corrupt.

$id_{1/2}$-abort-security: Either all honest parties get output, or they agree on a set of parties, at least half of which are corrupt.
Full Security

- Assume we have $id_{1/2}$-abort-secure protocol for general functions

- ECSS share inputs

- Run a $id_{1/2}$-abort-secure protocol to obtain ECSS shares of outputs

  - If abort/error, **eliminate** the identified set (who reshare their shares among active players). Repeat.

- If no abort, send shares for reconstruction

  - Note: honest majority maintained among active parties
Identifiable-Abort

Achieving identifiable abort requires additional redundancy so that intermediate shares in the multiplication step can be checked for consistency.

If check fails, can be attributed to a set of two players, at least one of whom is corrupt.
## Summary: General SFE

<table>
<thead>
<tr>
<th></th>
<th>Unlimited Corruption (using OT)</th>
<th>Honest Majority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Passive</td>
<td>“GMW” protocol</td>
<td>BGW protocol</td>
</tr>
<tr>
<td>Active</td>
<td>Unfair</td>
<td>Fair</td>
</tr>
<tr>
<td></td>
<td>via AMD circuits</td>
<td>via ECSS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>via identification &amp; elimination</td>
</tr>
</tbody>
</table>
Secure Multi-Party Computation

A Short Tutorial
Part II
Outline

- Definitions
  - Information-theoretic passive security
  - Simulation-based definitions
- Universal Composition
- Complexity Landscape
Information-Theoretic Passive Security

Say Alice & Bob have:
- inputs : X, Y
- outputs: A, B
- views in the protocol : U, V

Ideal views: (X,A) and (Y,B)

To be perfectly secure against curious Alice, U must be independent of (Y,B) given (X,A)

As a “Markov chain”: U — (X,A) — (Y,B)

Similarly for curious Bob: (X,A) — (Y,B) — V
Information-Theoretic Passive Security

- Ideal views: \((X,A)\) and \((Y,B)\)
- Real views: \(U\) and \(V\)

Security against curious Alice: \(U \rightarrow (X,A) \rightarrow (Y,B)\)

i.e., Alice could sample (simulate) \(U\) based on her ideal view \((X,A)\), so that it is consistent with the joint ideal views \((X,A,Y,B)\)

Environment's view: \(X,A,Y,B\), and whatever communication it has with the corrupt parties
Simulation-Based Security

Secure (and correct) if:

∀ s.t. output of is distributed identically in REAL and IDEAL
Simulation-Based Security

- Readily extends to active adversary and computational indistinguishability
- Corrupt parties route messages to/from the adversary
- Environment and Adversary are PPT
- This definition a.k.a. UC security
Hybrid Protocols

A “REAL” protocol in which parties access (another) IDEAL protocol
Concurrent Executions

∀ output of is distributed identically in REAL and IDEAL
Universal Composition

Replace protocol with which is as secure, etc.
Universal Composition

Replace protocol $\text{Env}$ with $\text{Env}$ which is as secure, etc.

World 1

World 3
Universal Composition

Replace protocol $\text{Alice} \leftrightarrow \text{Bob}$ with $\text{Alice} \leftrightarrow \text{Bob}^{\prime}$ which is as secure, etc.

Hope: resulting system is as secure as the one we started with.
Universal Composition

Start from world A (think "IDEAL")

Repeat (for any poly number of times):

For some 2 “protocols” (that possibly make use of ideal functionalities) I and R such that R is as secure as I, substitute an I-session by an R-session

Say we obtain world B (think “REAL”)

**UC Theorem**: Then world B is as secure as world A

Gives a modular implementation of the IDEAL world
Suppose we restrict to environments which interact with the rest of the system (parties, adversary) only prior to the start of, and after the end of the protocol

Original notion of security from 80’s and 90’s

Does not admit composition!
An example

Protocol:

Count down from 100

At each even round Alice announces whether her bid equals the current count; at each odd round Bob does the same.

Stop if a party says yes

Dutch flower auction

RECALL

Perfect Standalone Security
But doesn’t compose!
Attack on

Dutch Flower Auction

Alice and Bob are taking part in two auctions

Alice’s goal: ensure that Bob wins at least one auction and the winning bids in the two auctions are within ±1 of each other

Easy in the protocol: run the two protocols lockstep. Wait till Bob says yes in one. Done if Bob says yes in the other simultaneously. Else Alice will say yes in the next round.

Why is this an attack?

Impossible to ensure this in IDEAL!
Attack on Dutch Flower Auction

Alice’s goal: ensure that the outcome in the two auctions are within ±1 of each other, and Bob wins at least one auction.

Impossible to ensure this in IDEAL!

Alice could get a result in one session, before running the other. But what should she submit as her input in the first one?

- If a high bid, in trouble if she wins now, but Bob has a very low bid in the other session (which he must win).
- If a low bid (so Bob may win with a low bid), in trouble if Bob has a high bid in the other session.
UC Security

Very few 2-party functionalities have UC secure protocols (irrespective of computational assumptions)

- e.g., \( f(x, y) = x \)

Alternate option 1: Use a trusted set up

- OT would work, but too much to ask for?

Common reference string: produced by a trusted party once and for all

- CRS should not be used for anything else (anything in the environment using this string should be considered part of the adversary)!
Alternate option 2: “Angel-Based” Security

Allows the IDEAL adversary (simulator) access to unbounded computational power, while still guaranteeing universal composability.

Philosophy: In the IDEAL world, security guarantees are (should be) information-theoretic, so OK to allow unbounded adversary.

Then, all functionalities do have protocols under standard assumptions.
Cryptographic Complexity
Complexity w.r.t. MPC

We saw OT is complete for MPC

Any other functionality can be reduced to OT

Under all notions of reduction (passive-secure, or UC secure)

The Cryptographic Complexity question:

Can F be reduced to G (for different reductions)?

F reduces to G: will write $F \sqsubseteq G$

G **complete** if everything reduces to G

F **trivial** if F reduces to everything (in particular, to NULL)
Complexity w.r.t. MPC

- Several notions of $\sqsubseteq$
  - Passive, Standalone or UC
  - Information-theoretic or PPT
    - If PPT, also specify any computational assumptions used
- Will restrict to 2-party functionalities (mostly SFE)
- In particular, omitting honest majority security
PPT setting

- Under the assumption that there is a passive-secure protocol for OT (a.k.a. sh-OT)

- For passive & standalone security: all functionalities are trivial (and hence complete)!

- For UC security:
  - Recall, very few are trivial (e.g., \( f(x,y) = x \)) no matter what computational hardness assumptions used
  - UC trivial \( \iff \) Splittable, a simple characterization

- All other (finite) functionalities are complete!
IT Setting: Trivial Functionality

- Information-Theoretic Passive security
- Deterministic SFE: Trivial $\iff$ Decomposable
Decomposable Function

Decomposable

Undecomposable
IT Setting: Trivial Functionality

- Information-Theoretic Passive security
  - Deterministic SFE: Trivial $\iff$ Decomposable
  - Open for randomized SFE!
- Information-Theoretic Standalone security
  - Deterministic SFE: Trivial $\iff$ Uniquely Decomposable and Saturated
Decomposable Function

Decomposable

Not Uniquely Decomposable

Not Saturated

This strategy doesn’t correspond to an input
IT Setting: Trivial Functionality

- Information-Theoretic Passive security
  - Deterministic SFE: Trivial $\leftrightarrow$ Decomposable
  - Open for randomized SFE!

- Information-Theoretic Standalone security
  - Deterministic SFE:
    - Trivial $\leftrightarrow$ Uniquely Decomposable and Saturated

- Information-Theoretic UC security
  - Trivial $\leftrightarrow$ Splittable
PPT Setting: Completeness

- PPT Passive security and PPT Standalone security
  - Under sh-OT assumption, all functions are trivial — and hence all are complete too!

- PPT UC security
  - Recall, only a few (splittable) functionalities are trivial
  - Under sh-OT, turns out that every non-trivial functionality is complete
IT Setting: Completeness

- Information-Theoretic Passive security
- (Randomized) SFE: Complete ⇔ Not Simple
- What is Simple?
Simple vs. Non-Simple

Simple:
Each connected component is a biclique.

Non-Simple: (1,1)
IT Setting: Completeness

Information-Theoretic Passive security

(Randomized) SFE: Complete ⇔ Not Simple

What is Simple?

In the characteristic bipartite graph, each connected component is a biclique

(and, if weighted, $w(u,v) = w_A(u) \times w_B(v)$)
IT Setting: Completeness

- Information-Theoretic Passive security
  - (Randomized) SFE: Complete $\iff$ Not Simple

- Information-Theoretic Standalone & UC security
  - (Randomized) SFE: Complete $\iff$ Core is not Simple

- What is the core of an SFE?
  - SFE obtained by removing “redundancies” in the input and output space
Between Trivial & Complete?

- In the PPT setting, assuming sh-OT, there can be only one or two classes (two for UC security)
- In the IT setting, infinitely many levels!
- Question: Do these levels correspond to infinitely many complexity assumptions corresponding to which levels collapse in the PPT setting?
  - Probably not for UC security reductions
  - Only two such assumptions known so far: shOT & OWF
- Conjecture: Yes, for passive security reductions
Summary

- PPT, assuming sh-OT: 3 complexity classes, UC-trivial, UC-complete, all (= passive/standalone trivial/complete)
- IT: Infinitely many complexity classes. Several open problems.
- Computational assumptions related to collapse of classes in the PPT setting (so far OWF, shOT)
Today

- Brief overview of certain MPC protocols
- Several more that are not covered!
- Security Definitions, Universal Composition
- Minor and major variants not discussed
- Cryptographic Complexity
- Not covered: quantitative complexity
- Many more results. Even more open problems!